Project title: Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
Project acronym: MOSAICrOWN
Funding scheme: H2020-ICT-2018-2
Topic: ICT-13-2018-2019
Project duration: January 2019 – December 2021

D3.2
Preliminary Version of Tools for the Governance Framework

Editors: Aidan O Mahony (EISI)
Rigo Wenning (W3C)
Reviewers: Daniel Bernau (SAP SE)
Jonas Boehler (SAP SE)
Michele Mazzola (MC)
Megan Wolf (MC)

Abstract

This deliverable presents the preliminary version of tools for the governance framework. These tools provide the ability to ingest data into the MOSAICrOWN system. The tools were produced based on requirements introduced in D2.1 (“Requirements from the Use Cases”) and leverages the metadata model presented in D3.1 (“First version of the reference metadata model”). In order to accomplish the goal of producing the governance framework tools we present a state-of-the-art in the area of data governance tools within the scope of the requirements, examine potential implementation options, describe a deployment of the derived tool set, and we present research gaps identified during the course of this deliverable.

Type	Identifier	Dissemination	Date
Deliverable	D3.2	Public	2020.05.31

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 825333.

MOSAICrOWN Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	EMC Information Systems International	EISI	Ireland
3.	Mastercard Europe	MC	Belgium
4.	SAP SE	SAP SE	Germany
5.	Università degli Studi di Bergamo	UNIBG	Italy
6.	GEIE ERCIM (Host of the W3C)	W3C	France

Versions

Version	Date	Description
0.1	2020.05.05	Initial Release
0.2	2020.05.26	Second Release
1.0	2020.05.31	Final Release

List of Contributors

This document contains contributions from different MOSAICrOWN partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Aidan O Mahony (EISI)
Chapter 1: Introduction	Rigo Wenning (W3C), Aidan O Mahony (EISI)
Chapter 2: Requirements	Aidan O Mahony (EISI)
Chapter 3: State of the Art	Aidan O Mahony (EISI)
Chapter 4: Implementation Options	Aidan O Mahony (EISI), Matthew Keating (EISI), Rigo Wenning (W3C)
Chapter 5: Deployment	Aidan O Mahony (EISI), Shivam Chaurasia (EISI)
Chapter 6: Outlook and Further Research	Aidan O Mahony (EISI)
Chapter 7: Conclusions	Aidan O Mahony (EISI)

Contents

Executive Summary **9**

1 Introduction **11**
1.1 Purpose of this deliverable . . . 11
1.2 Scope of this deliverable . . . 11

2 Requirements **13**
2.1 Functional requirements . . . 13
2.2 Non-functional requirements . . . 14

3 State of the Art **15**
3.1 Data governance frameworks . . . 15
3.2 Extract, transform, load systems . . . 16
3.3 Big data: platforms and databases . . . 17
3.4 Policy storage . . . 18

4 Implementation Options **20**
4.1 Data formats . . . 20
4.1.1 JavaScript Object Notation . . . 20
4.1.2 Resource Description Framework . . . 20
4.1.3 JavaScript Object Notation for Linked Data . . . 21
4.1.4 NGSI-LD . . . 21
4.1.5 Other formats . . . 22
4.2 Web/Metadata Standards . . . 22
4.2.1 Web Ontology Language . . . 22
4.2.2 Dublin Core . . . 22
4.2.3 Open Data Protocol . . . 22
4.2.4 International Data Spaces Association - Reference Architecture Model . . . 22
4.2.5 Hierarchical Data Format version 5 . . . 23
4.3 Uniform Resource Identifier schemes . . . 23
4.3.1 Uniform Resource Names . . . 23
4.3.2 Magnet links . . . 23
4.3.3 InterPlanetary File System paths . . . 23
4.3.4 RFC 6920 - Naming Things with Hashes . . . 24
4.3.5 Hash URI . . . 24
4.3.6 Slash URI . . . 24
4.3.7 Internationalized Resource Identifiers . . . 24
4.3.8 Universally unique identifier . . . 24

5 Deployment **26**
5.1 High level design . . . 26
5.1.1 Configuration information . . . 27
5.2 Ingestion data . . . 27
5.2.1 ICV data . . . 28
5.2.2 Electric vehicle metadata . . . 28
5.2.3 Policy data . . . 29
5.2.4 Connecting data and metadata to policy . . . 29
5.3 MOSAICrOWN ingestion flow . . . 30
5.3.1 Jolt transform . . . 31
5.3.2 Addition of context and UUID . . . 31
5.4 Requirements verification . . . 32

6 Outlook and Further Research **41**
6.1 Data model extensions . . . 41
6.2 Query execution . . . 41
6.3 Ingestion data wrapping and sanitization . . . 42
6.4 Data storage . . . 42
6.5 Data analytics . . . 42
6.6 MOSAICrOWN configuration . . . 43

7 Conclusions **44**

Bibliography **45**

List of Figures

2.1 The MOSAICrOWN structure (ingestion) 13

4.1 ICV data in RDF format 25

5.1 The MOSAICrOWN ingestion high level design 27
5.2 Policy reference 29
5.3 The MOSAICrOWN NiFi ingestion processor 30
5.4 Metadata RDF model 33
5.5 Metadata OWL2 model 35
5.6 ICV data JSON Jolt transform 35
5.7 ICV input JSON 37
5.8 ICV data JSON 38
5.9 ICV metadata Jolt transform 39
5.10 ICV JSON-LD with id 39
5.11 ICV metadata JSON 40

6.1 Potential query execution 42

List of Tables

5.1 Configuration information 27
5.2 Electric vehicle data points 28
5.3 Electric vehicle metadata 29
5.4 NiFi data/metadata processors 31
5.5 NiFi policy processors 31
5.6 Requirements verification 32

6.1 Electric vehicle model comparison 41

Executive Summary

MOSAICrOWN aims to facilitate controlling and tuning of data protection by a data governance framework. This framework allows data owners to regulate in a simple, yet flexible and expressive way, access, visibility, and usage of their data. The framework will facilitate management, with clear identification of trust assumptions on the architecture and among parties, permitting controlled sharing and portability of data under data protection restrictions. The solutions will support secure and private sharing of data among organizations, each retaining control over their data. The result will then be a decentralized data governance model for multi-owner data platforms.

This deliverable presents the "Preliminary version of tools for the governance framework", comprising the initial tools which aim to satisfy the goals of WP3 ("Data governance framework") which is a M1-M34 work package. The results of this deliverable will be received by WP2 ("Requirements") and the tools will be applied to the Use Cases described in WP2, specifically T2.3 ("Testbed platform and deployment to Use Cases") and T2.4 ("Final testing and Use Case validation").

As other deliverables are presenting tools for sanitization and wrapping, we only consider governance and policy related tools and, as a preliminary step, we examine a subset of the three data life cycle phases. The three data life cycle phases for each use case are ingestion, storage, and data analytics.

Deliverable D3.2 presents tools for ingesting data, metadata, and policy into the MOSAICrOWN data market. Also, it presents the mechanism for connecting policy to metadata. The tools used in this deliverable will serve as the foundation of the tools applied to the Use Cases in WP2. The work will continue in the second phase of the project, enriching the solutions to fully account for the MOSAICrOWN objectives.

The deliverable is organized as follows. Chapter 1 introduces the purpose and scope of this deliverable. Chapter 2 details the functional and non-functional requirements which the tools address. Chapter 3 presents a state-of-the-art review of available governance tools. Chapter 4 presents further options aside from tools which are open to us when designing the governance framework tools. Chapter 5 analyzes the available tools and describes our deployed tools and furthermore, in this chapter, we illustrate how the requirements are satisfied by deploying our solution using data, metadata, and sample policies related to the use cases. Chapter 6 provides a summary of aspects still to be addressed and for which our tools need further development, and discusses their integration with other aspects of the project. Finally, Chapter 7 concludes the deliverable.

1. Introduction

MOSAICrOWN is driven by its use cases. Requirements were elaborated in D2.1, a first iteration of the requirements document. The deliverable D3.1 identified the metadata framework for the MOSAICrOWN use cases. In order to provide a data market that is viable from a business perspective, but also from a data protection perspective, data handling and data flows need to be controlled. One dimension of this control is metadata about rights to processing and other permissions attached to the actual data in the market. Data is *augmented* with those permissions in order to help later selection and processing for the data market. D3.1 had a first iteration of possible vocabularies on that and the MOSAICrOWN policy language will further contribute to augment the expressive power of that system.

1.1 Purpose of this deliverable

The purpose of this deliverable is to provide tools that are able to process the metadata and provide the necessary infrastructure to apply algorithms, permissions, restrictions and obligations in the course of the data way from the source to the data market. The tools will have to cope with data, metadata and a mix thereof. In order to be efficient, MOSAICrOWN wants to re-use a maximum of already existing state of the art tools. Where needed, those tools are adapted to the specific needs of the MOSAICrOWN use cases.

But MOSAICrOWN does not only have permissions attached to data. Some data in the use cases also must be transformed, sanitized, anonymized in order to make it usable for the data market. To make this easier, D3.1 introduced a simple data flow architecture where data flows from the source into a data lake and is subsequently selected and transformed to go into a data market. In this deliverable this basic model is also underlying the reflections on tooling. This deliverable will therefore not only address tools to handle data and metadata, it will also hint at the ways data will be funneled into and recovered from the data sanitization or which data to anonymize and which data to preserve.

The tooling described in this deliverable will then enable the application of the policy language, the sanitization, wrapping and transformation of data. This is the basic infrastructure that will create the pipes that finally fuel a successful data market.

1.2 Scope of this deliverable

The data governance framework involves different phases of the information life-cycle. The preliminary version of the tools focuses on data ingestion and addresses the problem of regulating data in the data lake and regulating the transformation. The three logic steps from D3.1 are taken into account: (i) data source discovery and ingestion (ii) data annotation and enrichment in the data lake (iii) data transformation (in the data lake or on the way to the data market). Each one

of those steps can contribute to the fulfilment of the requirements listed in Chapter 2. Some of the tools need chaining while others can be used standalone. Using linked data and its set of tools for certain tasks also helps to ensure combinability. The same generic tools can thus be re-used in a variety of different contexts. This is required in order to provide the data handling needed to constitute a working data market that is compliant with GDPR. Not only does that require a metadata infrastructure, but also tools to augment or reduce the amount of information present in the data.

2. Requirements

This chapter describes the requirements for the preliminary version of tools for the governance framework. The requirements are taken from deliverable D2.1 (“Requirements from the Use Cases”) which in turn were derived from Use Case 1 (UC1) (Protection of Sensitive Data in an Intelligent Connected Vehicle), Use Case 2 (UC2) (Data markets for analysis of financial data), and Use Case 3 (UC3) (Cloud-based data markets for privacy preserving Consumer analytics). The selection of the requirements was based on the scope as described in Chapter 1 Section 1.2 which is also illustrated in Figure 2.1. It is worth noting that most of the requirements were derived from UC1 therefore there is particular focus on this use case in this deliverable.

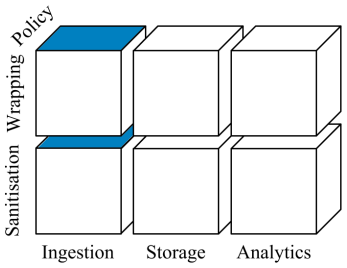

Figure 2.1: The MOSAICrOWN structure (ingestion)

2.1 Functional requirements

As described in D2.1, functional requirements define the expected behavior and functions of a system and are further detailed after initial identification within the system design phase. The functional requirements related to data ingestion are listed in this section and are categorized as per D2.1.

Data Ingestion

- REQ-UC1-DI1: MOSAICrOWN ingestion mechanism should support *close to source* deployment.
- REQ-UC1-DI2: Ingestion mechanism should support real-time stream data handling.
- REQ-UC1-DI3: Ingestion mechanism should support batch data handling.
- REQ-UC1-DI4: Ingestion mechanism should support different data types and formats, and account for structured and unstructured data.
- REQ-UC2-DI1: Ingestion mechanism should support batch data handling, which will enable the best level of automation.

- REQ-UC2-DI2: Ingestion mechanism should support different data types and formats (e.g., alphanumeric, integer, floating).
- REQ-UC2-DI3: All data feeds will be provided via governed and managed APIs.

Data Governance

- REQ-UC1-DG2: The platform should provide the capability for data providers to define data governance models for new data sets entering the platform.
- REQ-UC1-DG3: Data protection parameters for wrapping and sanitization should be configurable by the data owner.

Access Control Management

- REQ-UC1-AC7: Policies for data sets and platform users should be configurable by the data provider.
- REQ-UC2-AC4: All data accesses will be provided via governed and managed APIs.

Data Management

- REQ-UC1-DM3: Integrity of original data should be maintained separately in isolation, allowing an authorized entity to access data in the original state.

2.2 Non-functional requirements

Non-functional requirements address the operational needs of a system. These requirements, for example, address maintainability, usability or portability. The non-functional requirements relevant to data ingestion are listed in this section.

Data Controller

- REQ-UC2-DC1: Data Repository needs to be an independent organization and a separate data controller.
- REQ-UC2-DC2: External unique identifiers to be substituted with repository specific unique identifiers for use internally only within repository.

3. State of the Art

In this chapter we review the state of the art in the area of data governance and governance frameworks with the goal of identifying components suitable for deploying in the MOSAICrOWN data governance framework. In addition to data governance frameworks, we also need to consider ingestion technologies such as Extract, Transform and Load (ETL) tools, big data storage, and policy storage technologies.

3.1 Data governance frameworks

Data governance is defined as the processes, policies, standards, organization, and technologies required to manage and ensure the availability, accessibility, quality, consistency, auditability, and security of data in an organization [Pan10]. The tools examined in this section are amongst the most popular open source data governance frameworks available.

Apache Atlas is a data governance and metadata management platform [Qui18]. It facilitates ingest of metadata from various sources and it integrates with data from the Hadoop platform. Atlas also provides integration through messaging (Kafka based) or REST APIs and provides tracking enriched by business taxonomical metadata. Currently, Atlas supports ingesting and managing metadata from the following sources: HBase, Hive, Sqoop, Storm, Kafka.

ODPi Egeria is an open source project [1] dedicated to making metadata open and automatically exchanged between tools and data platforms, no matter which vendor they come from. The project provides an open metadata type system, frameworks, connectors, APIs, event payloads and interchange protocols to enable tools, engines and platforms to exchange metadata in order to get the best value from data and tools for a wide range of use cases. Egeria allows for deploying different types of Open Metadata and Governance (OMAG) servers each serving a specific purpose in different locations depending on the use case. Egeria stores and manages metadata through its governance servers which allow for extracting and synchronizing metadata with different technologies.

Octopai is a metadata management automation system [2]. Its use cases include data discovery, crawl, index, catalog, search and data lineage Compliance (i.e., GDPR, PII), data coalescing (relationship mapping) and business intelligence. Core Components include crawlers and ETL for various systems (SQL, SSIS, file, reporting platforms, Hadoop, etc.), cloud-hosted dashboard and

[1] https://www.odpi.org/projects/egeria
[2] https://www.octopai.com/

visualization. The strengths of this project include a broad set of connectors for databases and reporting platforms however there is a substantial up-front configuration to attach to back-end data systems and platforms.

Based on an evaluation of these governance frameworks, we believe that either Apache Atlas or ODPi Egeria might satisfy the requirements of MOSAICrOWN. As Octopai is a commercial framework it is not suitable for our usage. ODPi Egeria was incubated from Apache Atlas so there are some overlaps. For the final version of the data governance tools, these two frameworks are further evaluated.

3.2 Extract, transform, load systems

The software processes that facilitate the population of the data lake are commonly known as Extraction-Transformation-Loading (ETL) processes [Vas09]. ETL processes are responsible for (i) the extraction of the appropriate data from the sources, (ii) their transportation to a special purpose area of the data lake where they will be processed, (iii) the transformation of the source data and the computation of new values (and, possibly records) in order to obey the structure of the data warehouse relation to which they are targeted, (iv) the isolation and cleansing of problematic data, in order to guarantee that business rules and database constraints are respected and (v) the loading of the cleansed, transformed data to the appropriate relation in the warehouse, along with the refreshment of its accompanying indexes and materialized views. In this section, we inspect the most popular ETL systems with an aim of using one of them for ingesting data.

Talend by Gartner. This platform claims to collect/transform, govern and share data [KTN15]. The primary interface of this tool is called Open Studio. Open Studio Standalone application supports ETL operations, big data preparation, integration and quality.

Pentaho by Hitachi. This is a data integration and analytics platform which provides data operations for governance [PMSH16]. It claims to "unlock the maximum value from your data". It supports structured and unstructured data, with features designed for data acquisition and integration, quality, enrichment, and cleansing. Pentaho facilitates the transformation of raw data into a trusted resource for business insights that can be safely and securely shared with those who need them.

Jasper ETL by Jaspersoft. This is an open source data integration platform [MB15]. Jaspersoft ETL is a complete and ready-to-run ETL job designer with an extensive set of data integration capabilities. It extracts and transforms data from multiple systems and loads it into data stores optimized for reporting and analysis.

Apache Camel from the Apache Foundation. Camel is an open source integration framework that facilitates integration of various systems consuming or producing data [IA18]. It has many components that are used to access databases, message queues, or APIs. It runs as a standalone system, embedded as a library within Spring Boot, within application servers, and in the cloud. Camel supports around 50 data formats, allowing to translate messages in multiple formats, and with support from industry standard formats from finance, telco, and health-care.

Apache NiFi is a data flow platform [KLG19]. It is based on "NiagaraFalls" software developed by the NSA. NiFi as a data in motion technology that uses flow-based processing [YFJ17]. It enables data acquisition, basic event processing, and data distribution mechanism. NiFi gives organizations a distributed and resilient platform for building enterprise data flows [IZ18]. It provides the capability to accommodate diverse data-flows being generated by the connected world. NiFi enables connections among databases, big data clusters, message queues, and devices.

Node-RED is a visual programming tool that helps you connect your hardware, your apps and APIs in a useful way [BBB+18]. It provides a web-based editor that makes it possible to connect flows together and deploy them. Node-RED can be installed on Linux distros (Red Hat/Debian), Windows or even Raspberry and Android devices. Node-RED consists of a Node.js runtime on top of which you can create applications. Applications are created from a set of nodes that are available in Node-RED core and can be connected to form flows.

Microsoft SQL Server Integration Services (SSIS) is a component of the Microsoft SQL Server database software that can be used to perform a broad range of data migration tasks [KTN15]. SSIS is a platform for data integration and work flow applications. It features a data warehousing tool used for data extraction, transformation, and loading (ETL).

On reflection of the tools presented in this section, there are a number of tools which could satisfy the requirements of MOSAICrOWN. When the closed source tools are excluded the features presented by Apache NiFi and Node-RED in terms of extensibility and flexibility most closely align with MOSAICrOWN and both have the same license, i.e., Apache 2.0.

3.3 Big data: platforms and databases

The use of massive data requires the use of new technological tools for their capture from different sources and systems [AdR+17]. In this section we present systems used for storing and searching big data.

Hadoop Distributed File System (HDFS) is the primary data storage system used by Hadoop applications [SKRC10]. HDFS is an Apache Foundation project. It employs a NameNode and DataNode architecture to implement a distributed file system that provides high-performance access to data across highly scalable Hadoop clusters.

Apache Hive is a data warehouse system for data summarization and analysis and for querying of large data systems in the open-source Hadoop platform [HCG+14]. It converts SQL-like queries into MapReduce jobs for easy execution and processing of extremely large volumes of data.

MongoDB is a cross-platform document-oriented database program [Cho13]. Classified as a NoSQL database program, MongoDB uses JSON-like documents with a schema. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL).

Apache Storm is a free and open source distributed real-time computation system. Apache Storm makes it easy to reliably process unbounded streams of data, doing for real-time processing what Hadoop did for batch processing.

SAP-HANA is an in-memory, column-oriented, relational database management system developed and marketed by SAP SE [FCP+12]. Its primary function as a database server is to store and retrieve data as requested by the applications. In addition, it performs advanced analytics (predictive analytics, spatial data processing, text analytics, text search, streaming analytics, graph data processing) and includes extract, transform, load (ETL) capabilities as well as an application server.

HBase, an Apache open-source project, is a distributed fault-tolerant and highly scalable, column-oriented, NoSQL database built on top of HDFS [Geo11]. HBase is used for real-time read/write random-access to very large databases. In [Meh11], the authors conclude that HBase, as an open source alternative to traditional database management systems, is highly scalable, fault-tolerant, reliable, NoSQL, distributed database that operates on a cluster of commodity machines to handle large data volumes.

Based on the discussion in [AdR+17], Apache Hive and HDFS most closely aligns to our requirements as we will require big data analytics and Hive and HDFS are described in [AdR+17] as being most suitable for this purpose.

3.4 Policy storage

Policies guide the behavior of entities within the policy domain and have been used extensively in security, management and even network routing. Policies generally require application specific information to reason over, forcing researchers to create policy languages that are bound to the domains for which they were developed. The MOSAICrOWN policy language will be described in D3.3 (“First version of policy specification language and model”) however, for the purposes of this deliverable, we will illustrate policy related activities using a pre-existing policy language which uses the Web Ontology Language (OWL) to represent policy and as such we require a database suitable for storing OWL expressed policy.

Following research in the area of OWL based policy language [ORMG15], we note that graph-based databases are a popular method of storing data represented in OWL. Graph databases are based on graph theory from mathematics. Graphs are structures that contain vertices (which represent entities, such as people or things) and edges (which represent connections between vertices). Edges can have numerical values called weight.

In this section we present three such databases based on the literature review presented in [Ang12] and the survey presented in [DUG+10].

Apache Jena is an open source Semantic Web framework for Java [Jen15]. It provides an API to extract data from and write to RDF graphs. Jena provides support for OWL. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL, and includes a rule-based inference engine.

HypergraphDB is a general purpose, open-source data storage mechanism based on a powerful knowledge management formalism known as directed hypergraphs [Ior10]. In mathematics, a hypergraph is a generalization of a graph in which an edge can join any number of vertices. In contrast, in an ordinary graph, an edge connects exactly two vertices.

Neo4j is a graph database management system developed by Neo4j, Inc. Described by its developers as an ACID-compliant transactional database with native graph storage and processing [Mil13]. Neo4j is available both as a standalone server or an embeddable component.

Based on the work presented in [Ang12] and [DUG+10], we note that with regard to functionality, Apache Jena closely aligns with our desire to take advantage of semantic web standards.

4. Implementation Options

In this chapter we discuss the options available to this deliverable in terms of formats, standards, and methods of uniquely identifying the data being ingested into MOSAICrOWN.

4.1 Data formats

4.1.1 JavaScript Object Notation

JavaScript Object Notation (JSON) is an open standard file format used for data interchange. Text data is stored in human readable format consisting of key-value pairs. JSON formatting was originally developed by Douglas Crockford as an extension of JavaScript to handle object serialization [Oom14]. In 2013 JSON was standardized under EMCA-404i. RFC 8259 is the current version of the internet standard STD 90. The JSON format accepts all data types that are serializable.

4.1.2 Resource Description Framework

The Resource Description Framework (RDF) is a framework for expressing information about resources. Resources can be anything, including documents, people, physical objects, and abstract concepts, provided those resources are identified by a Uniform Resource Identifier (URI) [BLFM05] or Internationalized Resource Identifier (IRI) [DS05].

An RDF statement expresses a relationship between two resources. The subject and the object represent the two resources being related; the predicate represents the nature of their relationship. The relationship is phrased in a directional way (from subject to object) and is called in RDF a property. Because RDF statements consist of three elements, they are called triples.

```
UC1Car <is of type> JLR
UC1Car <has identifier> jooko4co
UC1Car <has driver> Bob
Bob <is> a natural person
UC1Car <collects> Vehicle location
Vehicle location <has> policy-chaug1di
Bob <has agreed> policy-chaug1di
```

In the context of the data market, RDF can be used to annotate arbitrary data provided it can be uniquely identified. From the above, we can conclude that Bob is driving UC1Car, that UC1Car is collecting location data and that, consequently, the system collects location data about Bob. To address GDPR requirements, use and other properties are defined in a policy and we can express that Bob has agreed to that policy. UC1 provides a context of data streaming from the vehicle into the data lake. RDF provides a common framework for expressing this data so it can

be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created. This means, car information can be streamed into the data lake without losing semantic and contextual information. Retrieving the IRI of UC1Car could then provide more data about the car, including links to other datasets for drivers, consumption, trips, etc. An automated process can then follow such links and aggregate data about these various things. Such uses of RDF are qualified as Linked Data in this document [BL06].

RDF has been standardized by the W3C and is the basis for other languages like Web Ontology Language (OWL), also used here. A good overview over RDF and the various specifications describing it can be found in the RDF-Primer [SR14] and as an example, Figure 4.1 shows Intelligent Connected Vehicles (ICV) data in XML-RDF format.

4.1.3 JavaScript Object Notation for Linked Data

JavaScript Object Notation for Linked Data (JSON-LD) is an extension of the JSON format developed to provide a method for encoding linked data within JSON. One of the goals of the JSON-LD format was to have a simple conversion process from JSON and JSON-LD making it easier for developers to implement. The format was originally developed by the JSON For Linking Data Community Group before being transferred to the RDF Working Group.

A context attribute is added to a JSON formatted file to provide links for each property to a specific ontology. This JSON-LD format can then easily be translated to RDF format allowing for a range of linked data related operations. An example of JSON-LD is presented here.

```
{
    "@context": {
        "name": "http://xmlns.com/foaf/0.1/name",
        "homepage": {
            "@id": "http://xmlns.com/foaf/0.1/workplaceHomepage",
            "@type": "@id"
        },
    "Person": "http://xmlns.com/foaf/0.1/Person"
    },
    "@id": "https://me.example.com",
    "@type": "Person",
    "name": "John Smith",
    "homepage": "https://www.example.com/"
}
```

4.1.4 NGSI-LD

NGSI-LD has been developed by FIWARE under the ETSI ISG CIM initiative. NGSI-LD was built in order to extend upon the JSON-LD format to better support linked data. The general structure of a NGSI-LD file is similar to that of JSON-LD however there are some changes to how the data is represented, NGSI-LD uses the format Entity (Instance), Property (Combination of attribute and value), Relationship to define data. The relationship attribute was implemented to

allow for better linked data representation, this allows for the association between two instances using URI pointers. NGSI-LD formatted files continue to use the @context attribute to define the ontology for that data.

4.1.5 Other formats

Comma separated values (CSV) is another format which will be ingested into MOSAICrOWN. The CSV file format has never been fully standardized however the format is relatively simple. A CSV file uses commas as field separator to order data. CSV has been a preferred way to store tabular data due to its compatibility with a wide range of data management tools.

4.2 Web/Metadata Standards

4.2.1 Web Ontology Language

Web Ontology Language (OWL) is an RDF based semantic web language for ontologies developed by the W3C. It represents semantics of documents and enables semantics to be consumed by machines, web applications and intelligent agents. It consists of classes, properties, individuals and data values to provide reasoning for semantic web documents. The primary exchange syntax for OWL 2 is RDF/XML (RDF Syntax) and this is indeed the only syntax that must be supported by all OWL 2 tools. OWL 2 is designed to facilitate ontology development and sharing via the Web, with the goal of making web content more accessible to machines.

4.2.2 Dublin Core

Dublin Core Metadata Initiative (DCMI) is an initiative supporting innovation in metadata design and best practices. As a project of The Association for Information Science and Technology (ASIS&T), DCMI has created the Dublin Core specification, contributed to other community-created specifications, and is considered a trusted manager of metadata vocabularies, schemes and other artifacts for more than 20 years.

The Dublin Core specification is the most basic metadata specification. It consists of 15 elements for core semantic definitions and is often combined with various models of metadata to form more complex descriptions.

4.2.3 Open Data Protocol

Open Data Protocol (OData) is an OASIS standard that defines the best practices for building and consuming RESTful APIs. After initial development by Microsoft, OData became a standardized protocol of the OASIS OData Technical Committee (TC). TC participants include CA Technologies, Citrix Systems, IBM, Microsoft, Progress Software, Red Hat, SAP SE and SDL.

4.2.4 International Data Spaces Association - Reference Architecture Model

The International Data Spaces Association - Reference Architecture Model (IDS-RAM) focuses on secure and trustworthy data exchange patterns in the manufacturing domain. It consists of five layers to establish interoperability and three crosscutting perspectives for reaching its main target, namely to ensure end to end data sovereignty of the data owner as well as secure and standardized data exchange in business ecosystems.

4.2.5 Hierarchical Data Format version 5

The Hierarchical Data Format version 5 (HDF5) is an open source file format that supports large, complex, heterogeneous data.

HDF5 is a data model, library, and file format for storing and managing data. The model includes an abstract data model and an abstract storage model (the data format), and libraries to implement the abstract model and to map the storage model to different storage mechanisms. The HDF5 Library provides a programming interface to a concrete implementation of the abstract model. The library also implements the programming model of data transfer, an efficient movement of data from one stored representation to another stored representation.

4.3 Uniform Resource Identifier schemes

A Uniform Resource Identifier (URI) is a string of characters that unambiguously identifies a particular resource. To guarantee uniformity, all URIs follow a predefined set of syntax rules, but also maintain extensibility through a separately defined hierarchical naming scheme (e.g., http://). Each URI begins with a scheme name that refers to a specification for assigning identifiers within that scheme. As such, the URI syntax is a federated and extensible naming system wherein each scheme's specification may further restrict the syntax and semantics of identifiers using that scheme. The URI generic syntax is a superset of the syntax of all URI schemes.

4.3.1 Uniform Resource Names

The Internet Engineering Task Force (IETF) informational RFC 1737 laid out functional requirements for Uniform Resource Names (URNs). Uniform Resource Names (URNs) are intended to serve as persistent, location-independent, resource identifiers.

4.3.2 Magnet links

Magnet is a URI scheme that defines the format of magnet links, a de facto standard for identifying files (URN) by their content, via cryptographic hash value rather than by their location. Magnet URIs consist of a series of one or more parameters, the order of which is not significant, formatted in the same way as query strings that ordinarily terminate HTTP URLs. The most common parameter is "xt" ("exact topic"), which is generally a URN formed from the content hash of a particular file, e.g.,

$magnet:?xt = urn:btih:c12fe1c06bba254a9dc9f519b335aa7c1367a88a$

4.3.3 InterPlanetary File System paths

InterPlanetary File System (IPFS) objects can be traversed with a string path API[Ben14]. Paths work as they do in traditional UNIX filesystems and the Web. The Merkle DAG links make traversing it easy. Note that full paths in IPFS are of the form:

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>
# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

IPFS is used in P2P and blockchain applications [CLLZ17].

4.3.4 RFC 6920 - Naming Things with Hashes

This scheme defines a set of ways to identify a thing (a digital object in this case) using the output from a hash function. It specifies a new URI scheme for this purpose, a way to map these to HTTP URLs, and binary and human-speakable formats for these names. An example of this URI scheme is presented below

$i:///sha-256;UyaQV-Ev4rdLoHyJJWCi11OHfrYv9E1aGQAlMO2X\_Q$

4.3.5 Hash URI

A HashURI is a URI with a hash mark ("#") in it (technically, a URI with a fragment identifier), specifically one used to denote something which is not a traditional web page.

Use of HashURIs is intended to help “WhenBrowsableAndUnambiguousCollide”. It is often presented in contrast to SlashURI, via the debate HashVsSlash.

4.3.6 Slash URI

A SlashURI is a URI without a hash mark ("#") in it (technically, a URI without a fragment identifier), especially one used in RDF to denote something which cannot be viewed in a browser. This term is in contrast to HashURI; both are ways of handling the situation when “WhenBrowsableAndUnambiguousCollide”. The terms sound nice together as HashVsSlash.

4.3.7 Internationalized Resource Identifiers

Internationalized Resource Identifiers (IRI’s) is a standard which was developed to extend the character set of Uniform Resource Identifiers (URI’s). IRI’s were first defined by the Internet Engineering Task Force in 2005 RFC 3987vii. While URI’s are limited to a subset of ASCII characters, IRI’s may contain any Unicode character. Other than this distinction, the function of URI’s and IRI’s are similar. Both use a hierarchical naming scheme to uniformly identify resources which can easily be extended. IRI’s are also backwards compatible allowing for them to be used in systems which only support the URI format. This is done by mapping the IRI to a URI representation.

4.3.8 Universally unique identifier

RFC 4122 describes how Universally Unique IDentifiers [LMS05] (UUIDs) are specified. A UUID is 128 bits long and can guarantee uniqueness across space and time. One of the main reasons for using UUIDs is that no centralized authority is required to administer them. As a result, generation on demand can be completely automated, and used for a variety of purposes.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns="https://uri.etsi.org/ngsi-ld/default-context/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ngsi-ld="https://uri.etsi.org/ngsi-ld/"
>
  <Vehicle rdf:about=
    "file:///base/data/home/apps/s%7Erdf-translator/
              2.408516547054015808/e97de6b6-5a4a-443c-8979-21c990a59de0">
    <vehiclePlateIdentifier>
      <ngsi-ld:Property rdf:nodeID="N7886dc8dcc634412954034c200a35a4c">
        <ngsi-ld:hasValue>BMW-7447</ngsi-ld:hasValue>
      </ngsi-ld:Property>
    </vehiclePlateIdentifier>
    <vehicleIdentificationNumber>
      <ngsi-ld:Property rdf:nodeID="N64c2e10fce964d26b5be7a042cc32f8c">
        <ngsi-ld:hasValue>7CKRXPHSZE</ngsi-ld:hasValue>
      </ngsi-ld:Property>
    </vehicleIdentificationNumber>
    <refVehicleModel>
      <ngsi-ld:Property rdf:nodeID="N28dd0d0109114b969d47f5ba0f562e13">
        <ngsi-ld:hasValue>Mercedes-Benz EQC</ngsi-ld:hasValue>
      </ngsi-ld:Property>
    </refVehicleModel>
    <color>
      <ngsi-ld:Property rdf:nodeID="N60844b75b49b48519bebb7f6522efa62">
        <ngsi-ld:hasValue>orange</ngsi-ld:hasValue>
      </ngsi-ld:Property>
    </color>
    <category>
      <rdf:Description rdf:nodeID="N427fd1fa742f4c0896b4395245c4a561">
        <ngsi-ld:hasValue>private</ngsi-ld:hasValue>
      </rdf:Description>
    </category>
    ...
  </Vehicle>
</rdf:RDF>
```

Figure 4.1: ICV data in RDF format

5. Deployment

In this chapter we describe our chosen options and deployed solution. Our solution makes use of available open source software augmented with custom data flows and use case specific transforms. Based upon the review of the available tools in Chapter 3, we decided to use Apache Jena as the graph-based database to store metadata and policy information, HDFS to store unstructured data, Apache Hive to facilitate structured data, and Apache NiFi for data ingestion processing. We also illustrate how our deployment satisfies the requirements outlined in Chapter 2.

5.1 High level design

Based on the tools and technologies detailed in Chapter 3 and Chapter 4 and with the knowledge of the data formats which the use cases use, we chose tools which were both versatile with regards to data formats, e.g., storage using HDFS for unstructured data and Apache NiFi for data flow management. We leveraged the Apache Software Foundation for the majority of these tools as there is both a rich development environment available and considerable interoperability which can be taken advantage of. In this section and we present the following high-level design based on the following components:

- `Ingestion data`: Data from UC1 will be used to demonstrate the preliminary version of tools for the governance framework. This data is in JSON format
- `Ingestion technology`: Apache NiFi for ingesting data both in batch mode (via the "GetFile" NiFi processor) and streaming mode (via the "ListenHTTP" NiFi processor)
- `Splitting data`: Ingestion data will be split into data and metadata via a "JoltTransformJSON" NiFi processor
- `UUID insertion`: Data and metadata will be assigned a UUID via the NiFi "Expression Language"
- `JSON conversion`: The conversion of JSON to JSON-LD will be carried out via a "ReplaceText" Regular Expression NiFi processor
- `Storage`: Storing data in HDFS will be carried out via the "PutHDFS" NiFi processor
- `Storage`: Storing metadata and policy data into the Jena database will be carried out via a standalone Java process

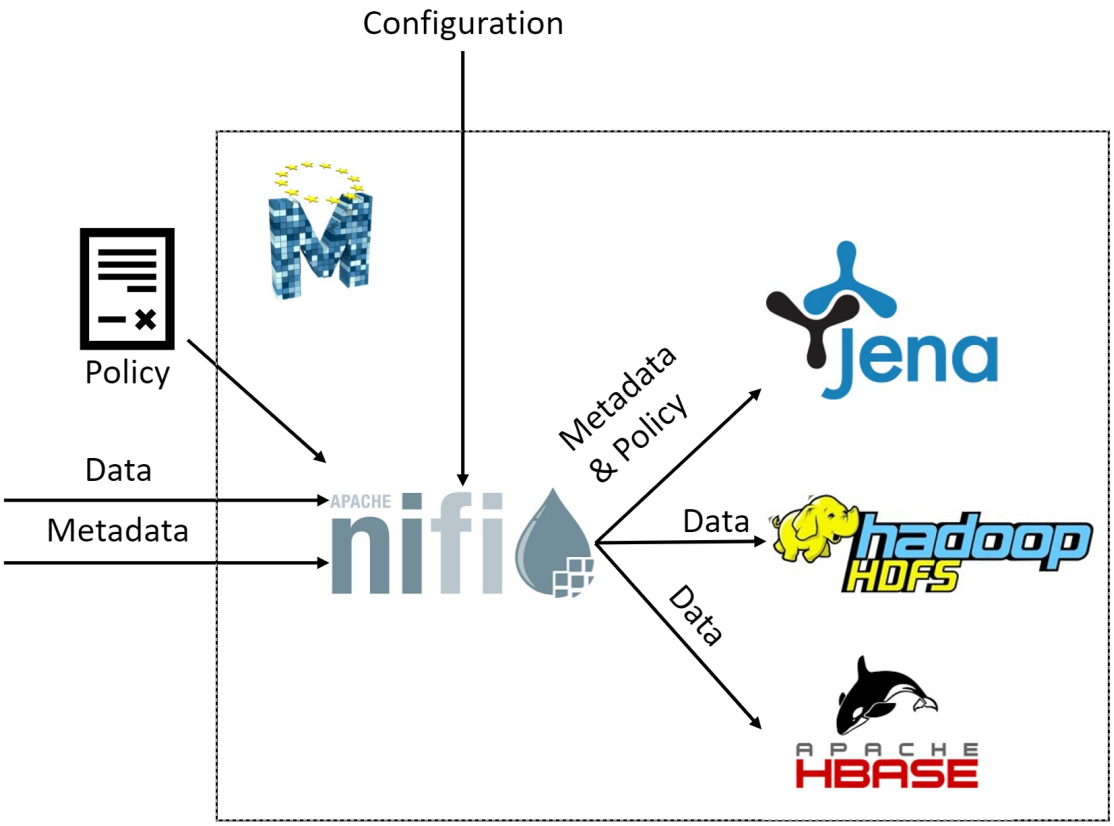

Figure 5.1: The MOSAICrOWN ingestion high level design

5.1.1 Configuration information

In this section we document the environment we deployed the tools on. As mentioned previously we leveraged open source software and these tools are presented in Table 5.1.

Component	Version	Description
CentOS	8.1.1911	Operating System
HDFS	3.2.1	Unstructured storage
Apache Jena	3.14.0	Graph based database
Apache NiFi	1.11.4	Dataflow system
Apache HBase	2.2.4	Table focused database

Table 5.1: Configuration information

5.2 Ingestion data

The ingestion data used to demonstrate this deployment is the data provided by UC1.

5.2.1 ICV data

An ICV generates sensor data and environmental data and combines them with identity data, for example, by identifying the driver or the car or both. The data points used to develop the platform which satisfies the requirements from Chapter 2 are shown in Table 5.2.

Category	Vehicle Data	Description
Fundamentals	Vehicle Status	Basic status information
	Capabilities	Vehicle capabilities
	Failure Message	Failure reason
	Firmware Message	Firmware version
	Historical State	Historical State
	Vehicle Status	Current status
Chassis	Charging State	Range, battery level, etc.
Diagnostics	Diagnostic State	Speed, battery voltage, etc.
	Maintenance	Battery service call date, etc.
	Usage	Last trip battery remaining, etc.
Infrastructure	Home Charger State	Charging power, solar charging, etc.
Points of Interest	Navi Destination	Navigation destination
	Vehicle Location	Current vehicle information
	Vehicle Time	Current vehicle time

Table 5.2: Electric vehicle data points

5.2.2 Electric vehicle metadata

A certain amount of metadata needs to be added to the data points read from the EV. These metadata are based on the FIWARE vehicle data model [SDM19] and are illustrated in Table 5.3. These metadata augment the ICV data at ingestion.

Title	Description
UID	Unique identifier
Type	Entity type. It must be equal to Vehicle
Source	A sequence of characters giving the source of the entity data
Name	Name given to this vehicle
Description	Vehicle description
Vehicle Type	Type of vehicle from the point of view of its structural characteristics
Category	Vehicle category(ies) from an external point of view
Location	Vehicle's last known location represented by a GeoJSON Point
Previous Location	Vehicle's previous location represented by a GeoJSON Point
Heading	Direction of travel of the vehicle
VIN	Vehicle Identification Number (VIN)
Vehicle Plate Identifier	An identifier or code displayed on a vehicle registration

Fleet Vehicle Id	Identifier of the vehicle in the context of the fleet of vehicles to which it belongs
Owner	Vehicle's owner
Date Modified	Last update timestamp of this entity
Date Created	Creation timestamp of this entity

Table 5.3: Electric vehicle metadata

5.2.3 Policy data

As detailed in Section 3.4, there will be MOSAICrOWN policy language used to enable the requirements for referring to metadata or facilitating data owners applying broader policy definitions. In this deliverable we used an OWL based representation sample policy to demonstrate the ingestion process.

5.2.4 Connecting data and metadata to policy

For MOSAICrOWN to enforce privacy preserving policy it is necessary to associate the data with policy. The simplest method is to supply a policy reference as part of the ingestion process. An example of how to do this is presented in Figure 5.2.

```
{
    "speed": {
        "type":"Property",
        "value":"2",
        "observedAt":"2020-04-23 15:01:03.855809"
    },
    "policy": {
         "type" : "Property",
         "value" : "policy-chaug1di"
    },
    "vehicleIdentificationNumber": {
        "type":"Property",
        "value":"2HRE0F4DW1"
    },
    "dateCreated": {
        "type":"Property",
        "value":"2020-04-23 15:01:03.855809"
    }
}
```

Figure 5.2: Policy reference

5.3 MOSAICrOWN ingestion flow

The NiFi ingestion flow is shown in Figure 5.3. All of the processors come with a standard NiFi distribution however, to implement the MOSAICrOWN requirements several transforms were created. These transforms were written using the JsOn Language for Transform (Jolt) transform language [AT18]. The NiFi processors used during ingestion are described Table 5.4. The NiFi data flow is saved as a "template" and this "template" can be version controlled.

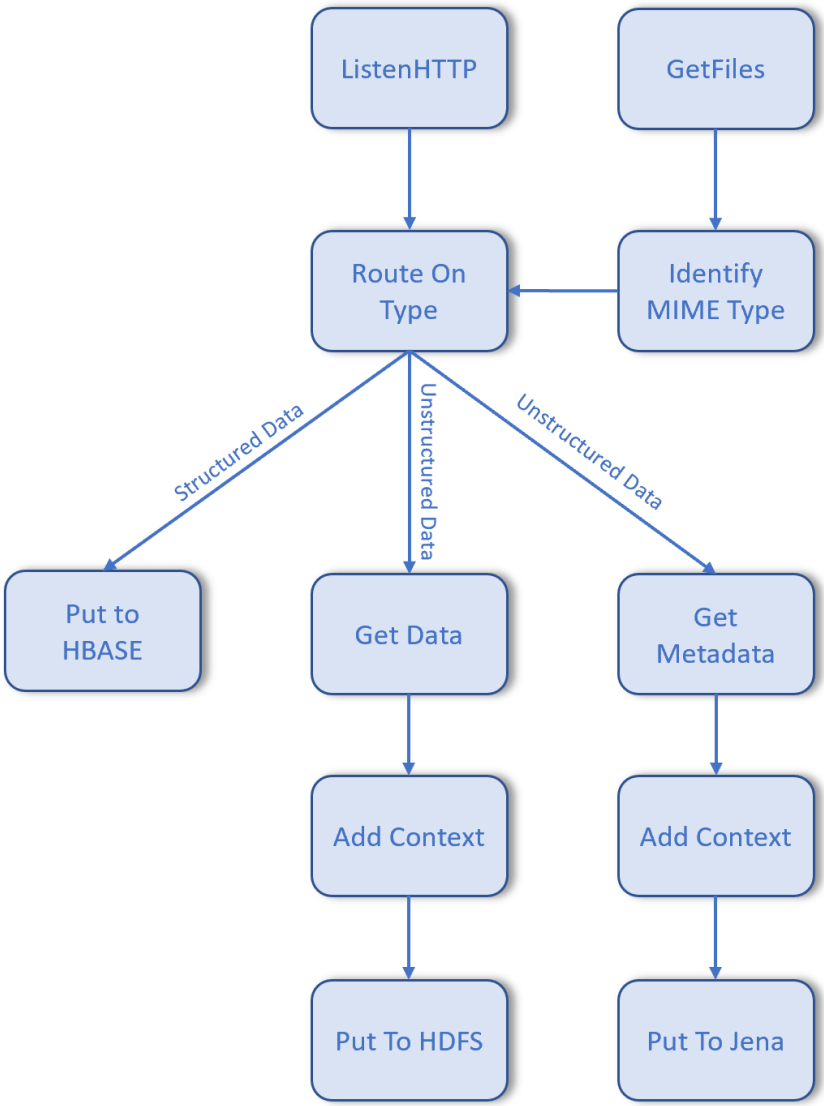

Figure 5.3: The MOSAICrOWN NiFi ingestion processor

Processor	Description
Listen HTTP	HTTP endpoint listening for data
Get Files	Consumes data from local file system
Route On Type	Routes data depending on content
Identify MIME Type	Determines MIME type of content
Get Data	Extracts data from incoming message

Get Metadata	Extracts metadata from incoming message
Add Context	Adds context information to JSON
Put to Jena	Places metadata into Jena ingestion folder
Put to HBASE	Inserts data into HBase database
Put to HDFS	Copies data to HDFS

Table 5.4: NiFi data/metadata processors

A separate flow exists for policy data using the processors listed in Table 5.5. The Jena ingestion procedure uses a standalone ingestion Java process started by an "ExecuteProcess" NiFi processor. Also required for writing metadata to the graph-based database is an "ontology model". We present an example of electric vehicle metadata in RDF in Figure 5.4 and also in OWL2 format in Figure 5.5.

Processor	**Description**
Listen HTTP	HTTP endpoint listening for data
Get File	Consumes data from local file system
Put Policy	Places OWL policy into Jena ingestion folder

Table 5.5: NiFi policy processors

5.3.1 Jolt transform

We leverage the Jolt transform language to split the incoming data into data and metadata. In this section we describe this process using the UC1 ICV data as an example. The data ingested in this example is the UC1 data and this data in JSON format is illustrated in Figure 5.7. Recall the data and metadata described in Figure 5.2 and Figure 5.3, we use two Jolt transforms to split the data into data and metadata. These Jolt transforms leverage the "shift" operation which reads values or portions of the input JSON tree and add them to specified locations in the output. An example of the data inputted to this transform is illustrated in Figure 5.7. The Jolt transform used is shown in Figure 5.6. The data JSON created from this operation is presented in Figure 5.8. Similarly, the Jolt transform used to create the metadata JSON is illustrated in Figure 5.9 and that output is shown in Figure 5.11.

5.3.2 Addition of context and UUID

The NiFi Expression Language allows us to access the attributes within the NiFi flowfile and manipulate these values. Combined with the "ReplaceText" processor in the NiFi data flow we can access the UUID of the flowfile and insert this as the id of the data and metadata linked data file. As example of the output of this step is shown in Figure 5.10. The output at this point is valid.

5.4 Requirements verification

In this section we evaluate our deployment against the requirements presented earlier in Chapter 2. This evaluation is shown in Table 5.6.

Requirement ID	Description	Comment
REQ-UC1-DI1	MOSAICrOWN ingestion mechanism should support *close to source* deployment	Deployment of ingestion tools can be made close to source
REQ-UC1-DI2	Ingestion mechanism should support real-time stream data handling	Real-time stream handling via HTTP server
REQ-UC1-DI3	Ingestion mechanism should support batch data handling	Batch data handling via NiFi processor
REQ-UC1-DI4	Ingestion mechanism should support different data types and formats, and account for structured and unstructured data	Can be supported by "RouteOnContent" and Jolt transforms
REQ-UC2-DI1	Ingestion mechanism should support batch data handling, which will enable the best level of automation	See REQ-UC1-DI3
REQ-UC2-DI2	Ingestion mechanism should support different data types and formats (e.g., alphanumeric, integer, floating)	JSON supports multiple data types and formats
REQ-UC2-DI3	All data feeds will be provided via governed and managed APIs	NiFi template version controlled
REQ-UC1-DG2	The platform should provide the capability for data providers to define data governance models for new data sets entering the platform	NiFi is extensible and can support alternative models
REQ-UC1-DG3	Data protection parameters for wrapping and sanitization should be configurable by the data owner	Supported via metadata
REQ-UC1-AC7	Policies for data sets and platform users should be configurable by the data provider	Supported via OWL ingestion
REQ-UC2-AC4	All data accesses will be provided via governed and managed APIs in their original state	See REQ-UC2-DI3
REQ-UC2-DC1	Data Repository needs to be an independent organization and a separate data controller	This can be supported if deployment of ingestion tools not close to source
REQ-UC2-DC2	External unique identifiers to be substituted with repository specific unique identifiers for use internally only within repository	Done for UC1, supported via Jolt transforms

Table 5.6: Requirements verification

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:j.0="https://uri.etsi.org/ngsi-ld/"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:j.1="http://www.w3.org/1999/02/22-rdf-syntax-ns/"
    xmlns:vehicle="https://uri.fiware.org/ns/data-models/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/heading"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/category"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/refVehicleModel"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/vehiclePlateIdentifier"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/dateCreated"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/speed"/>
  <owl:Class rdf:about="https://uri.etsi.org/ngsi-ld/location"/>
  <owl:Class rdf:about="https://uri.fiware.org/ns/data-models/vehicle"/>
  <rdf:Property rdf:about="https://uri.etsi.org/ngsi-ld/hasValue"/>
  <owl:DatatypeProperty rdf:about="https://uri.etsi.org/ngsi-ld/observedAt"/>
  <owl:DatatypeProperty rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns/type"/>
  <owl:DatatypeProperty rdf:about="https://uri.etsi.org/ngsi-ld/coordinates"/>
  <vehicle:speed rdf:about="https://uri.fiware.org/ns/data-models/">
    <j.1:type>id</j.1:type>
    <rdf:type rdf:resource="https://uri.etsi.org/ngsi-ld/location"/>
    <rdf:type rdf:resource="https://uri.fiware.org/ns/data-models/dateCreated"/>
    <rdf:type rdf:resource="https://uri.fiware.org/ns/data-models/refVehicleModel"/>
    <j.0:coordinates>http://www.w3.org/2001/XMLSchema/double</j.0:coordinates>
    <rdf:type rdf:resource="https://uri.fiware.org/ns/data-models/category"/>
    <j.0:observedAt>https://uri.etsi.org/ngsi-ld/DateTime</j.0:observedAt>
    <j.0:hasValue>https://uri.etsi.org/ngsi-ld/hasValue</j.0:hasValue>
    <rdf:type rdf:resource="https://uri.fiware.org/ns/data-models/heading"/>
    <j.1:type>https://uri.fiware.org/ns/data-models/Vehicle</j.1:type>
    <j.1:type>https://uri.etsi.org/ngsi-ld/GeoProperty</j.1:type>
    <j.1:type>https://uri.etsi.org/ngsi-ld/Property</j.1:type>
    <rdf:type rdf:resource="https://uri.fiware.org/ns/data-models/vehicle"/>
  </vehicle:speed>
</rdf:RDF>
```

Figure 5.4: Metadata RDF model

```
Prefix(ns0:=<http://www.w3.org/1999/02/22-rdf-syntax-ns/>)
Prefix(ns1:=<https://uri.etsi.org/ngsi-ld/>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)

Ontology(
Declaration(Class(ns1:location))
Declaration(Class(<https://uri.fiware.org/ns/data-models/category>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/dateCreated>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/heading>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/refVehicleModel>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/speed>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/vehicle>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/vehicleIdentificationNumber>))
Declaration(Class(<https://uri.fiware.org/ns/data-models/vehiclePlateIdentifier>))
Declaration(DataProperty(ns0:type))
Declaration(DataProperty(ns1:coordinates))
Declaration(DataProperty(ns1:observedAt))
Declaration(NamedIndividual(<https://uri.fiware.org/ns/data-models/>))
Declaration(AnnotationProperty(ns1:hasValue))

#############################
#   Named Individuals
#############################

# Individual: <https://uri.fiware.org/ns/data-models/>
# (<https://uri.fiware.org/ns/data-models/>)
AnnotationAssertion(ns1:hasValue <https://uri.fiware.org/ns/data-models/>
    "https://uri.etsi.org/ngsi-ld/hasValue")
ClassAssertion(ns1:location <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/category>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/dateCreated>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/heading>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/refVehicleModel>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/speed>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/vehicle>
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/vehicleIdentificationNumber>
```

```
    <https://uri.fiware.org/ns/data-models/>)
ClassAssertion(<https://uri.fiware.org/ns/data-models/vehiclePlateIdentifier>
    <https://uri.fiware.org/ns/data-models/>)
DataPropertyAssertion(ns0:type <https://uri.fiware.org/ns/data-models/>
    "https://uri.etsi.org/ngsi-ld/GeoProperty")
DataPropertyAssertion(ns0:type <https://uri.fiware.org/ns/data-models/>
    "https://uri.etsi.org/ngsi-ld/Property")
DataPropertyAssertion(ns0:type <https://uri.fiware.org/ns/data-models/>
    "https://uri.fiware.org/ns/data-models/Vehicle")
DataPropertyAssertion(ns0:type <https://uri.fiware.org/ns/data-models/> "id")
DataPropertyAssertion(ns1:coordinates <https://uri.fiware.org/ns/data-models/>
    "http://www.w3.org/2001/XMLSchema/double")
DataPropertyAssertion(ns1:observedAt <https://uri.fiware.org/ns/data-models/>
    "https://uri.etsi.org/ngsi-ld/DateTime")
)
```

Figure 5.5: Metadata OWL2 model

```
[
  {
    "operation": "shift",
    "spec": {
      "speed": "speed",
      "location": "location",
      "heading": "heading",
      "vehicleIdentificationNumber" : "vehicleIdentificationNumber",
      "dateCreated": "dateCreated"
    }
  }
]
```

Figure 5.6: ICV data JSON Jolt transform

```
{
  "type": "Vehicle",
  "category": {
    "value": [
      "private"
    ]
  },
  "refVehicleModel": {
    "type": "Property",
    "value": "Mercedes-Benz EQC"
  },
  "speed": {
    "type": "Property",
    "value": "76",
    "observedAt": "2020-04-23 15:00:55.219231"
  },
  "location": {
    "type": "GeoProperty",
    "value": {
      "type": "Point",
      "coordinates": [
        26.86907998942955,
        52.51859813773131
      ]
    }
  },
  "heading": {
    "type": "Property",
    "value": "236"
  },
  "color": {
    "type": "Property",
    "value": "orange"
  },
```

```
  "feature": {
    "type": "Property",
    "value": [
      {
        "value": "navi"
      }
    ]
  },
  "vehiclePlateIdentifier": {
    "type": "Property",
    "value": "BMW-7447"
  },
  "vehicleIdentificationNumber": {
    "type": "Property",
    "value": "7CKRXPHSZE"
  },
  "dateCreated": {
    "type": "Property",
    "value": "2020-04-23 15:00:55.219231"
  }
}
```


Figure 5.7: ICV input JSON

```
{
    "speed": {
        "type":"Property",
        "value":"2",
        "observedAt":"2020-04-23 15:01:03.855809"
    },
    "location": {
        "type":"GeoProperty",
        "value": {
            "type":"Point",
            "coordinates": [
                62.00553672280386,
                66.75066647791179
            ]
        }
    },
    "heading": {
        "type":"Property",
        "value":"326"
    },
    "vehicleIdentificationNumber": {
        "type":"Property",
        "value":"2HRE0F4DW1"
    },
    "dateCreated": {
        "type":"Property",
        "value":"2020-04-23 15:01:03.855809"
    }
}
```


Figure 5.8: ICV data JSON

```
[
  {
    "operation": "shift",
    "spec": {
      "type": "type",
      "category": "category",
      "refVehicleModel": "refVehicleModel",
      "color": "color",
      "feature": "feature",
      "vehiclePlateIdentifier": "vehiclePlateIdentifier",
      "vehicleIdentificationNumber": "vehicleIdentificationNumber",
      "dateCreated": "dateCreated"
    }
  }
]
```

Figure 5.9: ICV metadata Jolt transform

```
{
    "@context": [
        "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context.jsonld"
    ],
    "id": "e97de6b6-5a4a-443c-8979-21c990a59de0",
    ....
}
```

Figure 5.10: ICV JSON-LD with id

```
{
    "type": "Vehicle",
    "category": {
        "value": [
            "private"
        ]
    },
    "refVehicleModel": {
        "type": "Property",
        "value": "Mercedes-Benz EQC"
    },
    "color": {
        "type": "Property",
        "value": "orange"
    },
    "feature": {
        "type": "Property",
        "value": [
            {
                "value": "navi"
            }
        ]
    },
    "vehiclePlateIdentifier": {
        "type": "Property",
        "value": "BMW-7447"
    },
    "vehicleIdentificationNumber": {
        "type": "Property",
        "value": "7CKRXPHSZE"
    },
    "dateCreated": {
        "type": "Property",
        "value": "2020-04-23 15:00:55.219231"
    }
}
```


Figure 5.11: ICV metadata JSON

6. Outlook and Further Research

This deliverable provides a preliminary version of the governance tools and, as such, this chapter discusses both what areas of the governance framework require further development and research and shortcomings in the available tools and standards. The remaining development activities will be addressed in D2.4 (“Use Case prototypes”) and D3.4 (“Final tools for the governance framework”).

6.1 Data model extensions

The model used by UC1 is the FIWARE Vehicle model [Veh20] however this model has shortcomings when electric vehicles are required to be represented. Indeed, alternative models, such as the Schema.org Car model [MHS15], also have limitations, which we illustrate in Table 6.1. Important data points such as the estimated range of an EV or the battery level are not represented by all models.

MOSAICrOWN ICV data	**FIWARE Vehicle model**	**Schema.org Car model**
vin	vehicleIdentificationNumber	vehicleIdentificationNumber
modelName	refVehicleModel	model
name	name	name
licensePlate	vehiclePlateIdentifier	n/a
estimatedRange	n/a	n/a
batteryLevel	n/a	fuelCapacity
batteryVoltage	n/a	n/a

Table 6.1: Electric vehicle model comparison

6.2 Query execution

While this deliverable focused solely on ingestion, we also gave some consideration as to how query execution might be carried out. Our proposal is to facilitate query submission and query result access via a NiFi processor or processors. We illustrate how the data, metadata, and policies, might interact to supply users with query results in Figure 6.1. We will evaluate if the EU H2020 project “QROWD” [MIS+18] which developed a NiFi SPARQL processor.

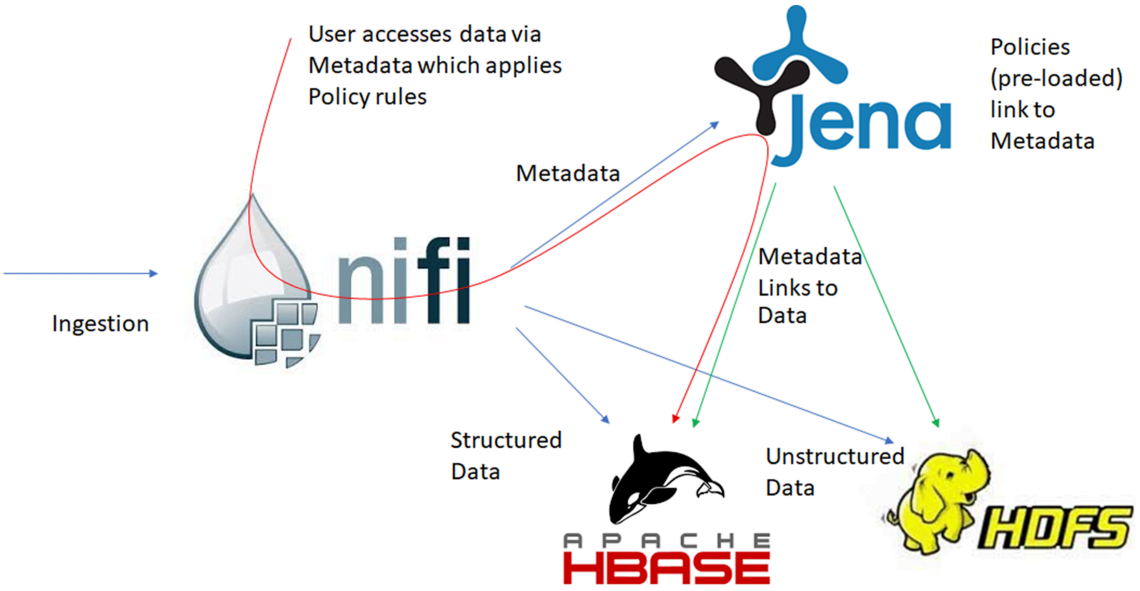

Figure 6.1: Potential query execution

6.3 Ingestion data wrapping and sanitization

The topics of data wrapping and data sanitization fell outside of the scope of this deliverable. The tools for data wrapping were developed in parallel for D4.1 (“First version of encryption-based protection tools”) and the tools for data sanitization were developed in parallel for D5.1 (“First version of data sanitization tools”). These tools will be integrated into the governance framework at a later stage.

6.4 Data storage

While we present solutions suitable for storing large volumes of data there are still requirements which need to be developed in the area of data storage. The data storage solution requires policy applications such as secure deletion, enforcing storage duration requirements, enforcing storage location requirements, e.g., permission to only store data in Europe. For HDFS the option of using a Hadoop daemon to accomplish some of these tasks is a possibility, for HBASE and Jena other options will be explored.

6.5 Data analytics

The data market has several requirements in the area of data analytics (both in the data market and facilitating analytics external to the data market). As described in D2.1 (“Requirements from the Use Cases”), the requirement topics include sanitization, enforcement of policy, and data movement services. One potential solution to satisfy these requirements in to leverage existing analytics engines and integrate such engine (or engines) into the governance framework. An example of a potentially suitable analytics engine is Apache Spark which complements our use of HDFS, Apache Jena, and Apache HBASE.

6.6 MOSAICrOWN configuration

The preliminary version of tools for the governance framework concerns itself solely with ingesting data from UC1. However, in order to fully implement all requirements the framework needs to facilitate configuration options such that the framework has context information on the data and metadata being ingested, e.g., what is the metadata? As MOSAICrOWN is multi-owner it will need to support multiple configurations. The exact mechanism that solves this problem remains to be solved.

7. Conclusions

This deliverable introduces the preliminary version of tools for the governance framework which forms the foundation of the platform which will satisfy the requirements of MOSAICrOWN. To accomplish this, we identified a set of functional and non-functional requirements suitable for implementation in the preliminary governance framework. These requirements were taken from the requirements supplied by all the industrial use cases. The deliverable, considering the data ingestion phase, focused on the data supplied by UC1.

As part of the design of the preliminary version of tools for the governance framework we conducted a state-of-the-art analysis of data governance tools including data governance frameworks, ETL tools, big data storage, and policy storage systems. Following this we detailed the options available to use in terms of ingestion formats, data and metadata standards available, and the schemes available to us in terms of unique reference identifier schemes.

From this catalogue of options and tools we identified the most suitable to our requirements and then deployed a solution based on Apache NiFi, Apache HBASE, and Apache Jena. This deployment was enhanced by custom NiFi processors developed for the preliminary version of tools for the governance framework.

From developing the preliminary version of tools for the governance framework we have the following remarks.

- For the use of linked data (such as the case in UC1) there are some gaps that need to be addressed in the available solutions.
- Facilitating query execution can be facilitated via the existing platform.
- Integration of data wrapping and data sanitization will occur in the next phase of the project.
- Application of policy within the data market, i.e., applied to the storage itself, might require development within the storage systems and controlled by a central MOSAICrOWN process.
- Data Analytics can be integrated by using existing open source frameworks, such as Apache Spark, with modifications and extensions to satisfy the requirements of all the use cases.

These remarks will form the basis to continue the work in the second phase of the project towards the release of the final tools for the governance framework.

Bibliography

[AdR+17] Susel Góngora Alonso, Isabel de la Torre Diez, Joel JPC Rodrigues, Sofiane Hamrioui, and Miguel Lopez-Coronado. A systematic review of techniques and sources of big data in the healthcare sector. *Journal of medical systems*, 41(11):183, 2017.

[Ang12] Renzo Angles. A comparison of current graph database models. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 171–177. IEEE, 2012.

[AT18] Elias Al-Tai. An evaluation of the expressive power and performance of json-to-json transformation languages. Master's thesis, KTH Royal Institute of Technology, 2018.

[BBB+18] Claudio Badii, Elefelious G Belay, Pierfrancesco Bellini, Mino Marazzini, Marco Mesiti, Paolo Nesi, Gianni Pantaleo, Michela Paolucci, Stefano Valtolina, Mirco Soderi, et al. Snap4city: a scalable iot/ioe platform for developing smart city applications. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 2109–2116. IEEE, 2018.

[Ben14] Juan Benet. IPFS - Content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561*, 2014.

[BL06] Tim Berners-Lee. Linked data-design issues, 2006. http://www.w3.org/DesignIssues/LinkedData.html.

[BLFM05] Tim Berners-Lee, Roy T Fielding, and Larry Masinter. Uniform resource identifier (uri): Generic syntax. Technical report, RFC Editor, 2005.

[Cho13] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, Inc., 2013.

[CLLZ17] Y. Chen, H. Li, K. Li, and J. Zhang. An improved P2P file system scheme based on IPFS and blockchain. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2652–2657, 2017.

[DS05] M. Duerst and M. Suignard. Internationalized resource identifiers (IRIs). Technical Report 3987, RFC Editor, January 2005.

[DUG+10] David Dominguez-Sal, Peter Urbón-Bayes, Aleix Giménez-Vanó, Sergio Gómez-Villamor, Norbert Martínez-Bazan, and Josep-Lluis Larriba-Pey. Survey of graph database performance on the hpc scalable graph analysis benchmark. In *International Conference on Web-Age Information Management*, pages 37–48. Springer, 2010.

[FCP+12] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner. SAP HANA database: data management for modern business applications. *ACM Sigmod Record*, 40(4):45–51, 2012.

[Geo11] Lars George. *HBase: the definitive guide: random access to your planet-size data*. O'Reilly Media, Inc., 2011.

[HCG+14] Yin Huai, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. Major technical advancements in apache hive. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1235–1246, 2014.

[IA18] Claus Ibsen and Jonathan Anstey. *Camel in action*. Manning Publications Co., 2018.

[Ior10] Borislav Iordanov. Hypergraphdb: a generalized graph database. In *International conference on web-age information management*, pages 25–36. Springer, 2010.

[IZ18] Haruna Isah and Farhana Zulkernine. A scalable and robust framework for data stream ingestion. *2018 IEEE International Conference on Big Data (Big Data)*, December 2018.

[Jen15] Apache Jena. A free and open source java framework for building semantic web and linked data applications, 2015. jena.apache.org.

[KLG19] Sang-Su Kim, Wang-Ro Lee, and Jun-Hui Go. A study on utilization of spatial information in heterogeneous system based on Apache NiFi. In *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1117–1119. IEEE, 2019.

[KTN15] Ranjith Katragadda, Sreenivas Sremath Tirumala, and David Nandigam. ETL tools for data warehousing: An empirical study of open source Talend studio versus Microsoft SSIS. In *ICWISCE'2015 International Conference on Web Information System and Computing Education, The 2nd World Congress on Computer Applications and Information*. Institute of Electrical and Electronics Engineers (IEEE), 2015.

[LMS05] Paul J. Leach, Michael Mealling, and Rich Salz. A universally unique identifier (UUID) URN namespace. RFC 4122, RFC Editor, July 2005. `http://www.rfc-editor.org/rfc/rfc4122.txt`.

[MB15] Nilesh Mali and Sachin Bojewar. A survey of ETL tools. *International Journal of Computer Techniques*, 2(5):20–27, 2015.

[Meh11] Mehul Nalin Vora. Hadoop-HBase for large-scale data. In *Proceedings of 2011 International Conference on Computer Science and Network Technology*, volume 1, pages 601–605, 2011.

[MHS15] Robert Trypuz Martin Hepp, Dominik Kuzinski and Karol Szczepañski. Car - schema.org type. https://schema.org/Car, May 2015. (Accessed on 04/23/2020).

[Mil13] Justin J Miller. Graph database applications and concepts with Neo4j. In *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA*, 2013.

[MIS+18] Eddy Maddalena, Luis-Daniel Ibáñez, Elena Simperl, Mattia Zeni, Enrico Bignotti, Fausto Giunchiglia, Claus Stadler, Patrick Westphal, Luís PF Garcia, and Jens Lehmann. Qrowd: Because big data integration is humanly possible. *Proceedings of the Project Showcase Track of KDD2018*, 2018.

[Oom14] Jeroen Ooms. The jsonlite package: A practical and consistent mapping between json data and r objects. *arXiv preprint arXiv:1403.2805*, 2014.

[ORMG15] Lorena Otero-Cerdeira, Francisco J Rodríguez-Martínez, and Alma Gómez-Rodríguez. Ontology matching: A literature review. *Expert Systems with Applications*, 42(2):949–971, 2015.

[Pan10] Zeljko Panian. Some practical experiences in data governance. *World Academy of Science, Engineering and Technology*, 62(1):939–946, 2010.

[PMSH16] Victor Parra, Azeem Mohammad, Ali Syed, and Malka N Halgamuge. Pentaho and Jaspersoft: A comparative study of business intelligence open source tools processing big data to evaluate performances. *International Journal of Advanced Computer Science and Applications*, 7(10):20–29, 2016.

[Qui18] Butch Quinto. Big data governance and management. In *Next-Generation Big Data*, pages 495–506. Springer, 2018.

[SDM19] Smart-Data-Models. Transportation harmonized data models, Oct 2019.

[SKRC10] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[SR14] A.Th. Schreiber and Y. Raimond. Rdf 1.1 primer. Technical report, W3C, June 2014.

[Vas09] Panos Vassiliadis. A survey of extract–transform–load technology. *International Journal of Data Warehousing and Mining (IJDWM)*, 5(3):1–27, 2009.

[Veh20] Vehicle - fiware-datamodels. https://fiware-datamodels.readthedocs.io/, Feb 2020. (Accessed on 02/04/2020).

[YFJ17] R. Young, S. Fallon, and P. Jacob. An architecture for intelligent data processing on iot edge devices. In *2017 UKSim-AMSS 19th International Conference on Computer Modelling Simulation (UKSim)*, pages 227–232, 2017.